The Visual Computer

# Algorithms for rendering realistic terrain image sequences and their parallel implementation

Gennady Agranov, Craig Gotsman

Computer Science Department, Technion – Israel Institute of Technology, Haifa 32000, Israel

We present algorithms for rendering realistic images of large terrains and their implementation on a parallel computer for rapid production of terrain-animation sequences. "Large" means datasets too large for RAM. A hybrid ray-casting and projection technique incorporates quadtree subdivision techniques and filtering using precomputed bit masks. Hilbert space-filling curves determine the image-pixel rendering order. A parallel version of the algorithm is based on a Meiko parallel computer architecture, designed to relieve dataflow bottlenecks and exploit temporal image coherence. Our parallel system, incorporating 26 processors, can generate a full color-terrain image at video resolution (without noticable aliasing artifacts) every 2 s, including I/O and communication overheads.

**Key words:** Parallel processing – Rendering – Terrain – Flight simulation

e-mail: {agranov|gotsman}@cs.technion.ac.il
*Correspondence to:* G. Agranov

## 1 Introduction

The speed/quality tradeoff for 3D terrain rendering is a major consideration for developers of visual flight simulation systems. To achieve true real-time flight simulation, such a system must produce at least 25 video-resolution color images per second. At the same time, these images should be of realistic quality and free of spatial and temporal aliasing and blurring effects. By "realistic" we mean that the images created are as similar as possible to what a pilot would see if he were positioned at the appropriate viewpoint, looking from the appropriate view angles at the terrain. For a description of the state of the art in these and related applications, see the survey by Cohen and Gotsman (1994).

Terrain rendering is achieved by combining two datasets – one containing information about terrain color (texture) from a vertical view angle (such as a satellite image) and the other containing information about terrain topography (elevation), supplying the third dimension. For the elevation dataset, the term digital terrain model (DTM) is widely used. The term digital elevation model (DEM) is also popular. In most applications the resolution of the DTM dataset is considerably lower than that of the texture – usually because it is difficult to obtain a DTM at the same resolution as the texture, but also because the texture contains more detail, essential for visual quality, than that present in the elevation data. (Our experience shows that ratio between DTM and texture resolution can be as high as 32 (in each dimension) without noticeable loss in visual quality.) In *polygon* systems, the terrain surface is reconstructed from the DTM as a $C^1$ triangle mesh, each triangle corresponding to many texture pixels. This constrasts with *voxel* systems (Cohen and Shaked 1993), in which the DTM is interpolated to produce values at a resolution identical to that of the texture. An elevation and RGB color are then associated with each such voxel (column). The collection of voxels is a $C^0$ reconstruction of the terrain surface.

The oblique perspective terrain image is created by mapping the texture onto the reconstructed terrain surface and projecting the surface onto a viewing plane, incorporating hidden surface elimination. Figure 1a shows a sample texture; Fig. 1b, the corresponding triangulated DTM; and Fig. 1d the result of mapping this texture to the triangle mesh surface. The geometric hidden

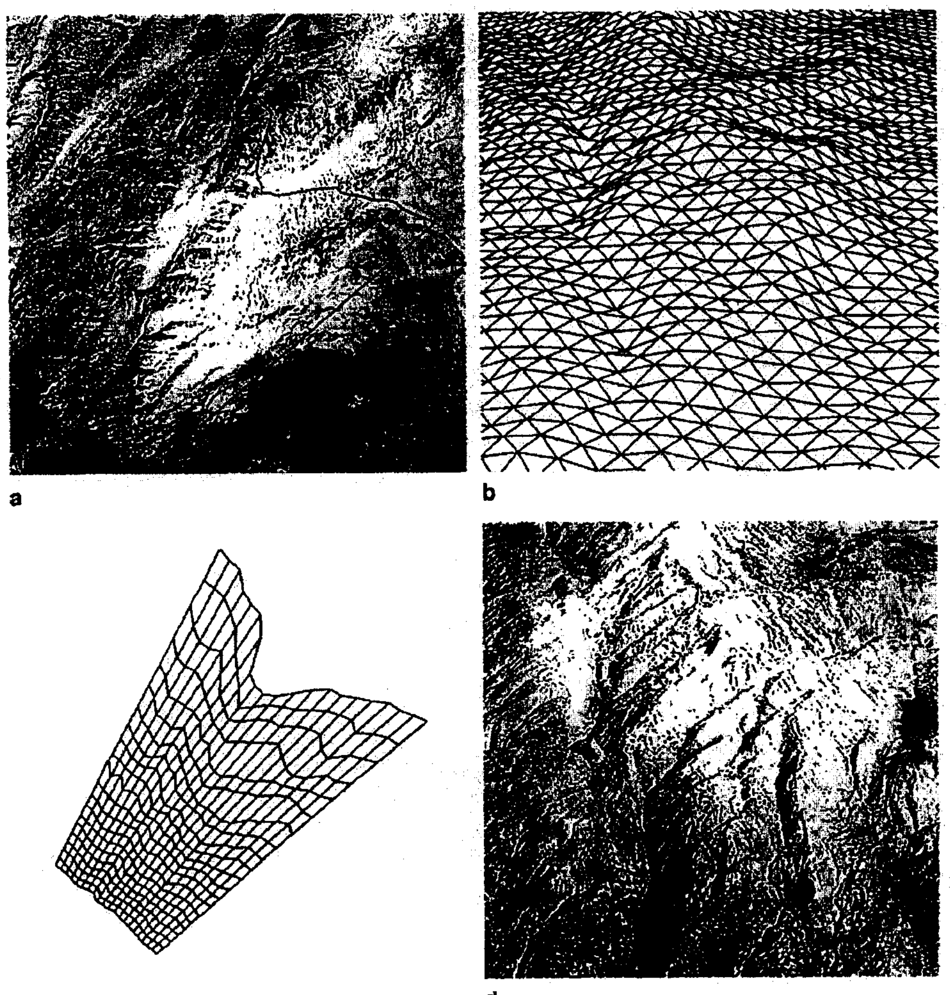


Fig. 1. a A sample texture (merged Spot + Landsat satellite image). The polygon indicates roughly the part of the texture contributing to the perspective view in d; b a 3D oblique view of the terrain corresponding to the area and view marked in a reconstructed from a subsampled DTM as a triangle mesh; c pixel polygons of a 20 × 20 pixel screen corresponding to the view b. Note how large some of the polygons are, implying that the filtering operation for that screen pixel might be costly; d oblique terrain image created by mapping the texture of a onto the terrain of b, or, alternatively, warping a according to c onto the screen pixels

surface elimination, essential for producing correct images, is the first potential bottleneck in the rendering.

In typical flight simulation scenarios, image *sequences*, over large areas, are required. The texture database, corresponding to this area, is usually very large and cannot be held in the main memory (RAM) of the computer. Sometimes, for views with very low pitch angles, even the texture required for just a single image cannot be held in RAM in full resolution. The performance of high-end graphics workstations, such as the SGI Reality Engine (Akeley 1993), relies on all the texture being present in RAM, and such work stations typically cannot handle such large terrain scenarios. In our system, like in some other 3D visual simulation systems (Folby et al. 1993; Geymayer et al. 1991; Kaba and Peters 1993; Leclerc and Lau 1994), the texture and DTM are stored on disk as a sequence of fixed-size squares, called tiles. At any given moment, only a small subset of these tiles are present in RAM. An important factor in any system design is the efficient management of such offline data, minimizing the amount of I/O incurred during the rendering of the sequence. This dataflow problem is a second potential bottleneck. It may be alleviated by special-purpose image-retrieval hardware over high-speed networks (Leclerc and Lau 1994). We address the problem by dynamic paging (Folby et al. 1993) of the data from external storage. In the case of slow interprocessor communication, a processor may also maintain its own private cache.

Rendering realistic terrain image *sequences* requires special care, because, even though each individual image of the sequence may appear reasonable, the display of the sequence may result in acute "flickering" and "shimmering", a manifestation of temporal aliasing. This can be alleviated, even eliminated, by correct filtering, in both spatial and temporal domains. The filtering, essentially weighted averaging of a large number of texture pixels, tends to be costly in time, becoming a third potential bottleneck. Almost all flight simulator systems (Folby et al. 1993; Leclerc and Lau 1994) use special-purpose graphics hardware for rendering, which provide minimal antialiasing, but even the 4×4 subpixel supersampling Reality Engine (Akeley 1993), one of the most advanced commercially available graphics engines, does not provide adequate filtering for this application.

In this paper we present efficient algorithms to alleviate these three bottlenecks, namely, hidden surface elimination, dataflow, and filtering. These algorithms were created as part of an ongoing effort to develop efficient techniques for high-quality flight simulation. At the current state of technology, it seems that only a hardware platform incorporating parallel processing has any chance of achieving this goal. Towards this end, we are experimenting with a Meiko parallel computer, containing 28 i860 processors with 8 MB of RAM each. The size of our terrain is a 256 km × 512 km rectangle of 10 m × 10 m resolution Spot-Landsat-merged satellite imagery, and a 80 m × 80 m resolution DTM.

We present a parallel architecture and implementation of our algorithms on a Meiko computer, designed specifically to cope with the three bottlenecks just mentioned. We have not yet achieved real-time performance, but the rendering speed of the system increases linearly with the number of processors.

The paper is organized as follows: Sects. 2–4 describe our methods of alleviating the bottlenecks. Section 5 describes our parallel implementation. We conclude in Sect. 6.

## 2 Hidden surface elimination

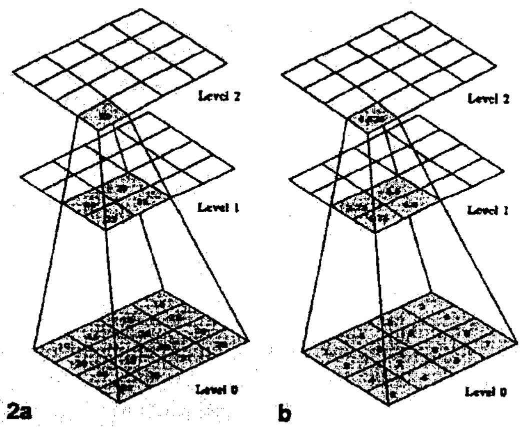
There are two main approaches in the literature to hidden surface elimination in terrain triangle meshes:

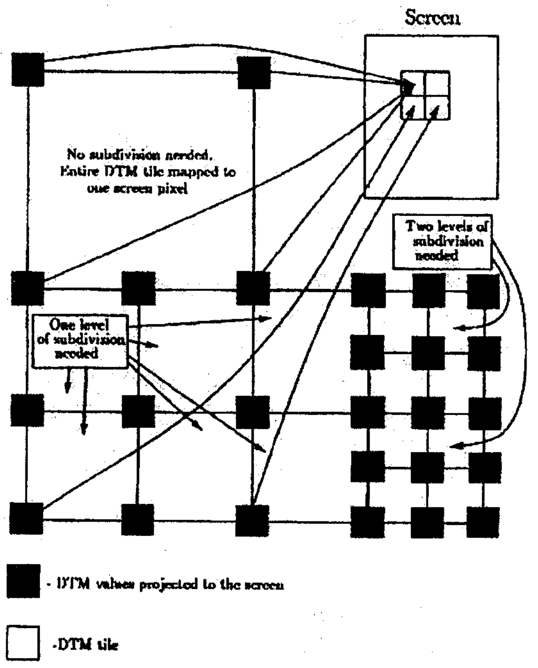
- *Ray casting.* A ray is cast from the viewpoint through each screen pixel center and its first intersection with the terrain is calculated. This may be thought of as *bringing the screen to the terrain.* The main advantage of ray casting is that there are no computational expenses for the hidden or clipped parts of terrain, but even so, existing ray casting algorithms (Musgrave 1988) are extremely slow. Attempts to increase the speed by various types of acceleration methods [hierarchical (Cohen and Shaked 1993) or parametric (Paglieroni and Petersen 1992)] improve the situation significantly, but still do not achieve real-time speeds on high-end workstations. An advantage, however, of the ray casting method is that parallelization is almost trivial. Even so, to achieve a reasonable rendering time (e.g., less than 1 s) for an image of video resolution, a massively parallel computer (Vezina and Robertson 1991) or special parallel ray casting hardware [for example, Pitot et al. (1989)] is required.
- *Projection.* The 3D triangles are projected onto the 2D screen, and hidden surfaces eliminated with a *Z*-buffer algorithm. This can be thought of as *bringing the terrain to the screen.* The main advantage of this approach is its speed, but there are also expenses for the hidden/clipped parts of the terrain, which do not contribute to the final image. In large terrains, it is hard to tell in advance which triangles will contribute to the final image. The traditional "brute force" solution of projecting *all* scene geometry and letting the software/hardware eliminate the redundant ones is obviously impractical in this scenario.

We use a hybrid ray casting and projection technique. Ray casting is performed only for the pixels along the screen border. The intersection points of these rays with the terrain form the vertices of a polygon, bounding the DTM data tiles (containing triangles), which suffice to render the scene. *Only triangles contained in these tiles are projected.* The rest of the triangles can be eliminated ("culled"). In cases where the horizon is present in the image, for which the bounding polygon is potentially infinite, it is truncated to the borders of the texture database. This approach is more complicated than the standard method of culling the tiles up to some artifical distance limit [e.g., 26 km (Folby et al. 1993)], but provides higher-quality background pixels.

For the ray casting, we use the hierarchical ray casting technique (Cohen and Shaked 1993). It is an accelerated version of traditional incremental ray casting, originally designed to run on voxel terrains. The data structure supporting this algorithm is the *elevation pyramid* (Fig. 2a). At its base lies the original elevation database. A value at any higher level is the maximum of four adjacent elevations in the level below it, forming a quadtree structure. Using an elevation pyramid, we have adapted the hierarchial ray casting method to run on triangle meshes in a straightforward manner. Our algorithm provides the exact position on the terrain of the intersection of a ray with the triangle mesh.

After ray casting the border pixels, the triangles in the bounded tiles should be projected onto the image plane. Obviously, the farther away from the foreground we are, the less detailed DTM information we need. This is manifested in the size of the projection of a single triangle sometimes being less than a screen pixel. However, the technique of using DTM tiles or various resolutions (Folby et al. 1993; Geymayer et al. 1991; Leclerc and Lau 1994), chosen according to the distance from the viewpoint, causes "earthquake" effects, during transition between the discrete resolution levels during animation. This is unacceptable for high-quality rendering. Instead, we use tiles at a fixed level of resolution, and the following recursive approach: given such a tile, we project its four corner elevations. If they fall within the same screen pixel, we are done. If not, we subdivide the tile to four equal-area quadrants, and deal recursively with the quadrants. Subdivision is




Screen
No subdivision needed. Entire DTM tile mapped to one screen pixel
Two levels of subdivision needed
One level of subdivision needed
- DTM values projected to the screen
-DTM tile

3

Fig. 2. Hierarchical data structures used for rendering: a elevation pyramid; b texture pyramid

Fig. 3. The terrain subdivision technique used for rendering. ■ digital terrain model (DTM) values projected to the screen; □ DTM tile

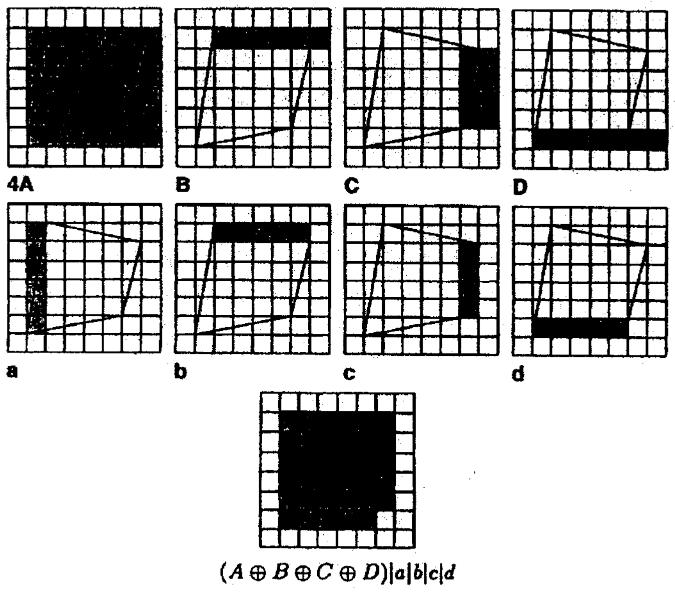
terminated either when all the corners of the subtile are mapped to the same screen pixel – and then the $Z$-buffer value of that pixel is set to the minimum of the four values – or when the quadrant reduces to two terrain triangles. Note that this rendering technique is an approximation, for if a DTM tile contains singular (very narrow and tall) spikes of subpixel width, they will be lost in the process. From our experience with natural terrains, this does not occur. The subdivision process is illustrated in Fig. 3. When the process is terminated for the second reason, we use a modified $Z$-buffer scheme, with the rational interpolation technique (Heckbert 1990), to scan-convert the projections of these triangles, obtaining the ($x$, $y$) coordinates on the terrain, and in texture space, of each screen pixel corner [a similar procedure is used in the texture-mapping hardware of the SGI Reality Engine (Ackeley 1993)]. This is needed for accurate texture filtering, on which we elaborate in the next section.

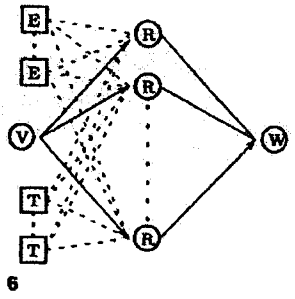
## 3 Filtering

For high-quality antialiased rendering, every screen pixel should be considered not a point, but a small square of finite area. The color value assigned to the screen pixel during rendering should be a weighted average of all texture pixels projected onto this square.
To locate these pixels, map the four corners of the screen pixel to the terrain. This defines a *pixel polygon* in texture space. All texture pixels within this polygon participate in the average. Figure 1c shows the pixel polygons associated with a $20 \times 20$ pixel image of the terrain of Fig. 1d. In effect, the vertices of these pixel polygons define a warp of texture space to image space, as these vertices must be mapped to a regular grid representing the corners of the screen pixels. Note that our description of rendering as a texture warp implies that, ultimately, *all* texture pixels in the pixel polygon contribute to the final image, even those that are occluded in reality. We found this approximation to be quite reasonable, the occluded texture pixels affecting only a few image pixels, and, in fact, "emphasizing" terrain silhouettes. There are two ways to obtain the average of the texture pixels within a pixel polygon:

- *Direct Filtering*. Exhaustive averaging of the texture pixels in the pixel polygon: this gives an accurate result, but when the pitch angle is small or viewpoint very high, the pixel polygon may be arbitrarily large. In this case, a large number of texture pixels must be averaged, which is time consuming. This is compounded by the fact that not all the texture necessarily resides in RAM, so apart from CPU time, costly I/O may be required.
- *Filtering on a pyramid*. This is a simple way to approximate the result of direct filtering efficiently. A *texture pyramid* (Williams 1983) is constructed, so that a texture pixel at any level in this pyramid is the average of four values from the previous level, and covers four times the area (Fig. 2b). The base of the pyramid is the original texture database. All levels of this pyramid are also stored as tiles. All texture tiles are the same size in pixels, but cover areas of different size on the terrain, depending on the level. Filtering is performed by sampling the pyramid. Various techniques may be employed, depending on the quality of the approximation required [see Heckbert (1986) for a survey]. The simpler techniques use only one level of the pyramid, determined by the area of the pixel polygon, and the more sophisticated techniques interpolate between two adjacent levels. The latter is obviously more accurate, but also slower.

The direct filtering method is too costly and the accuracy of the crude pyramid approximation to a pixel polygon is not sufficient for rendering high-quality animation sequences. We employ a hybrid approach, filtering at one pyramid level, determined by the area of the pixel ploygon. We compute at a level *lower* than the standard pyramid filtering algorithms prescribe, so that the pixel polygon is contained in an $8 \times 8$ pixel texture square (a $16 \times 6$ pixel square can also be used for higher quality). Averaging the pixels of this square contained in the pixel polygon will result in a very accurate filtered value. To reduce the time of this costly filtering operation, we use the method of precomputed bit masks, as in the $A$-buffer algorithm (Carpenter 1984). This yields a (small) constant-time algorithm for calculating the final mask specifying all the pixels inside or on the border of the pixel polygon. Once this mask is known, we use the higher levels of the texture pyramid to calculate the filtered value efficiently, by quadtree decomposition of the bitmap (see Fig. 4). Our experiments show that for the $8 \times 8$





| | | | | |
|---|---|---|---|---|
| 0 | 1 | 2 | 0 | ...... |
| 1 | 2 | 0 | 1 | ...... |
| 2 | 0 | 1 | 2 | ...... |
| 0 | 1 | 2 | 0 | ...... |
| ⋮ | ⋮ | ⋮ | ⋮ | |

7

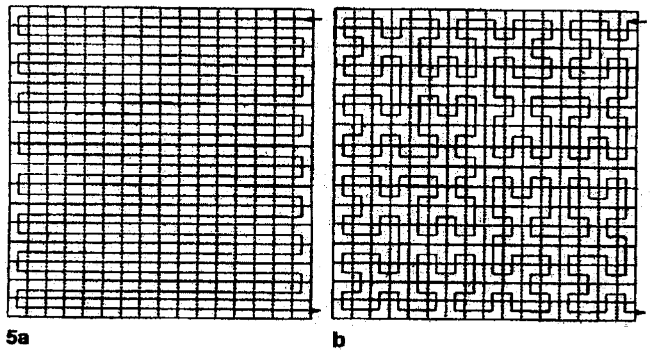


**Fig. 4.** High-quality texture filtering. The pixel polygon mask is a bitwise operation on the eight precomputed masks A, B, C, D, a, b, c, d, indexed by the vertices of the pixel polygon. Each of A, B, C, D represents the set of pixels to the "right" of a polygon edge, and a, b, c, d, the edge itself

**Fig. 5a, b.** Possible scan orders for pixel rendering: **a** raster; **b** Hilbert space-filling curve

**Fig. 6.** The architecture of our parallel rendering system on the Meiko computer

**Fig. 7.** Assignment of data tiles to three cache processors according to the mapping $M(i, j) = (i+j) \bmod 3$

square, filtering time was reduced to 20% of the time of the direct filtering algorithm, while use of a 16×16 pixel square reduced filtering time to 40%.

## 4 Dataflow management

The most time-consuming component of image sequence rendering in large terrain databases is

the I/O incurred by loading the texture tiles containing data needed for the rendering of a screen pixel; that is, filtering of the appropriate pixel polygon in texture space. Even though the tiles may be compressed [e.g. with JPEG (Wallace 1991)], a cache must be maintained explicitly to minimize unnecessary I/O. The image spatial coherence guarantees that the set of data tiles required for a pixel will have a significant intersection with the set of tiles required for an adjacent pixel.
One possible way to render the screen pixels while minimizing texture tile traffic is to scan (in any order) the screen pixels and determine for each a list of the texture tiles needed for its rendering. These lists are then united and sorted, so that each texture tile is loaded just once, and all needed calculations performed on it. We checked this approach, and discovered that it was slower than the one to be proposed next. This is because of the time overhead imposed by the sorting, and the accounting necessary to keep track of each texture pixel's contribution to each screen pixel. There is also a space penalty for saving the tile ID lists for all screen pixels.
The following method proved to be more efficient: render the screen pixels in a traversal order reflecting their proximity, thus increasing the hit ratio of the texture tile cache. Figure 5 shows two possible rendering orders: a suboptimal "raster" order, and a better "space-filling" curve with the attractive feature that points geometrically close to each other are usually not far apart along the curve. Consequently, most tiles do not have to be communicated again when we go from pixel to pixel. Sizes which are powers of two, such optimal Hilbert curves may be easily constructed [see, for example, Witten and Wyvill (1983) for details on this and other curves], and for sizes that are not powers of two, the construction of Perez et al. (1992) can be used.

## 5 The parallel system

### 5.1 Architecture and design

Our parallel system for terrain rendering is implemented on a Meiko computer containing 26 Intel i860 reduced instruction-set computer (RISC) processors. Each processor has 8 MB of RAM, which is relatively small for our application. Each processor also has 8 unidirection communication channels (with 1 M/s bandwidth), and the interprocessor connection topology can be configured by the user. The rendering is performed in a three-stage pipeline, the second stage consisting of a variable number of processors (see Fig. 6), each rendering a small subimage of the final image. Another two types of processors maintain texture and elevation data caches. The processors are divided into the following five categories, and connected through the virtual computing surface (VCS) of the computer:

- *V (view) – the main control unit.* This processor generates the view parameters for each image of the sequence. As the image is rendered in parts, the processor hierarchically ray casts the pixels along the borders of the subimages, and supplies a list of elevation and texture data tiles required for the rendering of each subimage to the appropriate R processor, as described in Sect. 2.
- *R (render) – the rendering processors.* These processors receive the view parameters from the V processor, render the subimage, as described in Sect. 3 and 4, and send the results to the W processor. The R processor has a small internal cache for texture and elevation tiles. If a required texture (elevation) tile is not in its cache, it is shipped in from the T(E) processor. Each R processor renders the *same* subimage for all frames in the sequence. This way, temporal coherence of the image sequence is exploited to maximize the hit ratio of its cache.
- *W (write) – the output processor* – receives rendered subimages from R processors, combines them to one final image, and stores this image.
- *T (texture) – a texture cache* – maintains a large cache of texture tiles, supplying them on demand to the R processors.
- *E (elevation) – an elevation cache* – maintains a large cache of elevation data tiles, supplying them on demand to the R processors.

Apart from the V and W processors, which are unique, there may be a variable number of the other types of processors. Obviously, a large number of R processors, which do the actual rendering, is recommended. If more than one T(E) processor is used, the load is distributed evenly between

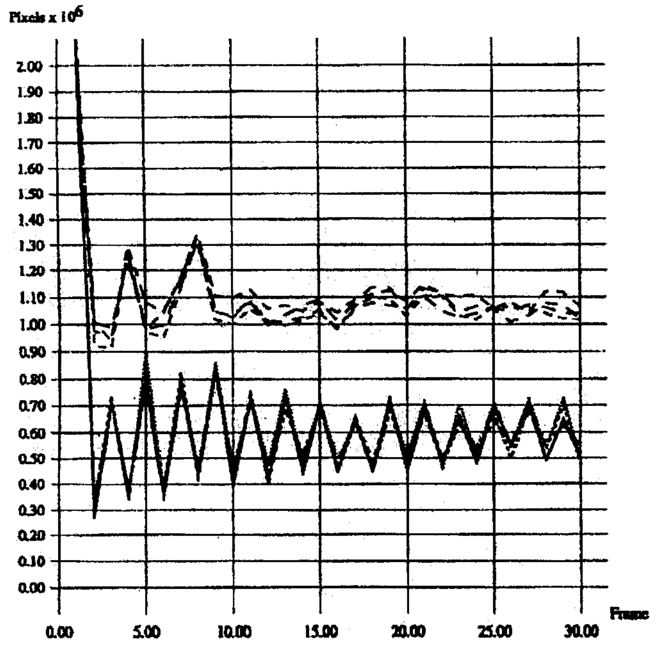


**Fig. 8.** The total number of texture pixels, communicated in tiles of 64 x 64 pixels, shipped from each of the four T processors to the nine R rendering processors. The top four curves are in the case of an image raster scan, the bottom in the case of a Hilbert scan. It is evident that the load is evenly distributed among the T processors in both cases. The speed of the flight is such that new tiles are required all the time

these $n$ processors. We ensure this by using the following addressing scheme: processor $T_k(E_k)$ $(0 \leq k < n)$ deals only with the $i, j$th tiles such that $(i+j) = k(\text{mod } n)$. Any R processor requiring a specific tile communicates with the appropriate T(E) processor, according to this rule. Figure 7 shows the assignment of the tiles of the database to three cache processors.

When an R processor encounters a situation in which a required elevation tile is not present in its own RAM, it sends a request to the E processor, and continues rendering other triangles. The tile is processed on arrival. Similarly, when a T or E processor encounters a situation in which a required tile is not in its cache, it initiates an I/O procedure, continuing the processing of other requests.

## 5.2 Performance

We have produced phase-alternate-line (PAL) video-resolution (576 × 768 pixels) image sequences with our parallel system, using an experimental configuration consisting of nine R, four T, and one E processors. The size of the texture tiles in the database is 64 × 64 pixels. Interprocessor communication overhead incurred by the system is measured by the number of pixels shipped to the R processors per rendered image. The efficiency of the parallel architecture is measured by the load balance between the R processors, and the load balance between the data cache T and E processors.

Figure 8 shows the total number of texture pixels, communicated in tiles of 64 × 64 pixels, from the four T to the nine R processors. To check the gain from using the Hilbert scan to render pixels, we compared this to the identical statistic obtained when a regular raster scan was used (also plotted on the graph). The graph shows that the spatial image coherence used by the Hilbert curve reduces the amount of communication required to approximately 55%. It can also be seen that in both the Hilbert and the raster-scan cases, the load is evenly distributed among the four T processors, confirming the efficiency of the tile-addressing scheme described in the previous section.

Figure 9 depicts the number of texture pixels actually *used* by each of the R processors during

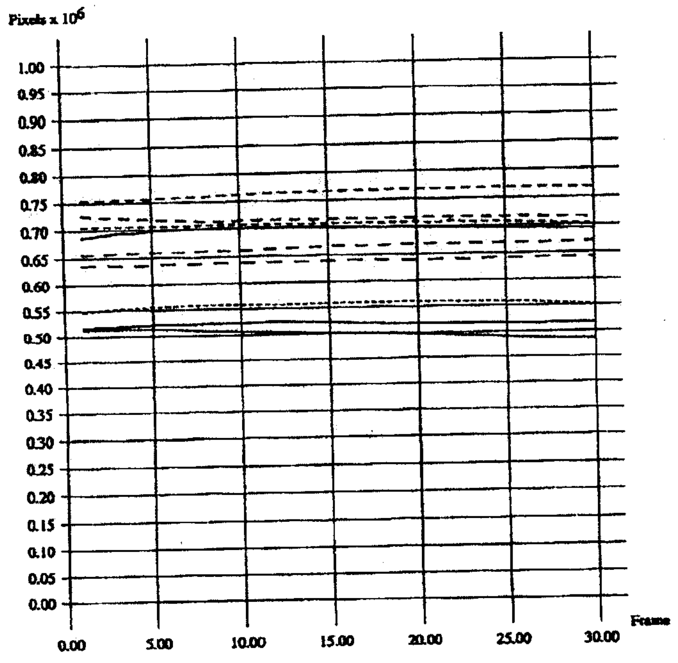


**Fig. 9.** The number of texture pixels actually used by each of the nine R processors when rendering a subimage

the rendering. The sum of these numbers (on the average 5.8 MB per frame) is larger than the number of *communicated* pixels (2.4 MB per frame), as the internal caches of the R processors supply more than half the required pixels.

Note that not *all* communicated pixels are necessarily used, as they are supplied in integral tile chunks, and some of the pixels in the tiles contributing to the image pixels along a subimage border may be redundant. For this configuration, the amount of overhead is typically 60% of the amount of nonredundant data. It could theoretically be reduced by using smaller tiles, but this would result in an increase of memory required to store and maintain the texture pyramid table in the R and T processors. We experimentally found the tile size of 64 × 64 pixels to be optimal. The fact that all nine curves of Fig. 9 are close together indicates that the rendering load is more or less evenly distributed among the R processors. A typical image pixel is a filtered value of approximately 13 texture pixels. The exact number per image pixel depends on the local scale of the image in the vicinity of that pixel relative to the hierarchy of scales in the texture pyramid.

When all 26 processors of the Meiko were used (a typical configuration would be 16 R, 6 T, 2 E, 1 V, and 1 W processors), we were able to achieve an almost optimal speedup in rendering time. This results in the production of a PAL-resolution, 24-bit, color image every 2 s, including I/O and communication overheads.

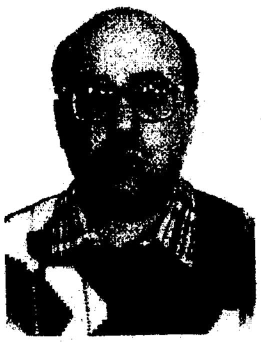
## 6 Conclusions

We have described efficient special-purpose algorithms for rendering terrain-image sequences. We have also described a parallel terrain-image-rendering system, using these algorithms to produce high-quality video-resolution imagery at high rates. Our long-term goal remains to achieve real-time visual flight simulation. We believe that this cannot be achieved on state-of-the-art graphics workstations without significant improvement of their ability to handle very large datasets. Parallel processing, however, does hold promise for this application, as we have demonstrated here. We continue to optimize the speed/quality tradeoff by optimizing our rendering algorithms and by identifying and alleviating bottlenecks in system throughput.

*Acknowledgements* We thank R. Cleave of Rohr Productions (Nicosia) for providing the satellite image data used in this work, and J. Hall of the Geological Survey of Israel (Jerusalem) for providing the DTM data. Partial support for C. Gotsman's work was provided by the Technion Vice President for Research Fund – New York Metropolitan Research Fund. G. Agranov's work is partially supported by a grant from the National Council for Research and Development – Israel Ministry of Science.

## References

Akeley K (1993) Reality Engine graphics. Comput Graph 27:109–116
Carpenter L (1984) The A-buffer, an antialiased hidden surface method. Comput Graph 18:103–108
Cohen D, Gotsman C (1994) Photorealistic terrain imaging and flight simulation, IEEE Comput Graph Appl 14:10–12
Cohen D, Shaked A (1993) Photorealistic imaging of digital terrains. Proceedings of Eurographics '93 373, UK, Eurographics, Blackwell, pp 363–373
Folby J, Zyda M, Pratt D, Mackey R (1993) NPSNET: hierarchical data structures for real-time three-dimensional visual simulation. Computers and Graphics 17:65–69
Geymayer B, Prant M, Muller-Seelich H, Tabatabai B (1991) Animation of landscapes using satellite imagery. Proceedings of Eurographics '91, Eurographics, Geneva, pp 437–446
Heckbert PS (1986) Survey of texture mapping (1986) IEEE Comput Graph. Appl 6:56–67
Heckbert PS (1990) Generic convex polygon scan conversion and clipping. In: Glassner A (ed) Graphics Gems I, Academic Press, San Diego, pp 84–86
Kaba J, Peters J (1993) A pyramid-based approach to interactive terrain visualization. Proceedings of the Symposium on Parallel Rendering, San Jose, ACM Press, New York, pp 67–70
Leclerc VG, Lau SQ (1994) Terra Vision: a terrain visualization system. Technical Report 540, SRI International, Menlo Park, Calif
Musgrave FK (1988) Grid tracing: fast ray tracing for height fields. Technical Report YALEU/DCS/RR-639, Yale University Department of Computer Science
Paglieroni DW, Petersen SM (1992) Parametric height field ray tracing Proceedings of Graphics Interface '92, Vancouver, Canadian Society for Human-Computer Interaction, pp 192–200
Perez A, Kamata S, Kawaguchi E (1992) Peano scanning of arbitrary size images. Proceedings of the 11th International Conference on Pattern Recognition, Hague, IEEE Computer Science Press, Los Alamitos
Pitot P, Duthen Y, Caubet R (1989) A parallel architecture for ray casting. Proceedings of Computer Graphics International '89, Leeds. Springer, Tokyo, pp 463–472
Vezina G, Robertson FK (1991) Terrain perspectives on a masively parallel SIMD computer. Proceedings of Computer Graphic International '91, M.I.T., Springer, Tokyo, pp 163–188
Wallace GK (1991) The JPEG still picture compression standard. Commun ACM 34:30–44
Williams L (1983) Pyramidal parametrics. Comput Graph 17:1–11
Witten AH, Wyvill B (1983) On the generation and use of space-filling curves. Software Pract Experience 13:519–525



GENNADY AGRANOV is a Research Fellow in the Computer Department at the Technion, Israel Institute, of Technology in Haifa, Israel. Dr. Agranov earned his PhD from the Leningrad Institute of Technology in 1989, and has lived in Israel since 1991. He is currently engaged in research in high-quality 3D rendering and animation.

CRAIG GOTSMAN is a Senior Lecturer in the Computer Science Department at the Technion, Israel Institute of Technology in Haifa, Israel. Dr. Gotsman earned his PhD from the Hebrew University of Jerusalem in 1991. His research interests are computer graphics and computational geometry, particularly the fields of rendering and animation. Dr. Gotsman is also a consultant at the Hewlett-Packard Israel Science Center in Haifa.

# Distributed Parallel Data Storage Systems: A Scalable Approach to High Speed Image Servers

*Brian Tierney (bltierney@lbl.gov),*
*William E. Johnston[1](wejohnston@lbl.gov),*
*Hanan Herzog, Gary Hoo, Guojun Jin, Jason Lee,*
*Ling Tony Chen*, Doron Rotem**

*Imaging and Distributed Computing Group and *Data Management Research Group*
*Lawrence Berkeley Laboratory[2]*
*Berkeley, CA 94720*

## Abstract

We have designed, built, and analyzed a distributed parallel storage system that will supply image streams fast enough to permit multi-user, "real-time", video-like applications in a wide-area ATM network-based Internet environment. We have based the implementation on user-level code in order to secure portability; we have characterized the performance bottlenecks arising from operating system and hardware issues, and based on this have optimized our design to make the best use of the available performance. Although at this time we have only operated with a few classes of data, the approach appears to be capable of providing a scalable, high-performance, and economical mechanism to provide a data storage system for several classes of data (including mixed multimedia streams), and for applications (clients) that operate in a high-speed network environment.

## 1.0 Introduction

In recent years, many technological advances have made possible distributed multimedia servers that will allow bringing "on-line" large amounts of information, including images, audio and video, and hypermedia databases. Increasingly, there also are applications that demand high-bandwidth access to this data, either in single user streams (e.g., large image browsing, uncompressible scientific and medical video, and multiple coordinated multimedia streams) or, more commonly, in aggregate for multiple users. Our work focuses on two examples of high-bandwidth, single-user applications. First, the terrain visualization application described below requires 300-400 Mbits/s of data to provide a realistic dynamic visualization. Second, there are applications in the scientific and medical imaging fields where uncompressed video (e.g. from typical laboratory monochrome video cameras that produce 115 Mbits/s data streams) needs to be stored and played back at real-time rates. In these example applications compression is not practical: in the case of terrain visualization, the computational cost of decompression is prohibitive; in the case of medical and scientific images, data loss, coupled with the possible introduction of artifacts during decompression, frequently precludes the use of current compression techniques. (See, for example, [7].)

Although one of the future uses of the system described here is for multimedia digital libraries containing multiple audio and compressed video streams, the primary design goal for this system is to be able to deliver high data rates: initially for uncompressed images, later for other types of data. Based on the performance that we have observed, we believe, but have not yet verified, that the approach described below will also be useful for the video server problem of delivering many compressed streams to many users simultaneously.

### Background

Current disk technology delivers about 4 Mbytes/s (32 Mbits/s), a rate that has improved at about 7% each year since 1980 [8], and there is reason to believe that it will be some time before a single disk is capable of delivering streams at the rates needed for the applications mentioned. While RAID [8] and other parallel disk array technologies can deliver higher throughput, they are still relatively expensive, and do not scale well economically, especially in an environment of multiple network distributed users, where we assume that the sources of data, as well as the multiple users, will be widely distributed. Asynchronous Transfer

1. Correspondence should be directed to W. Johnston, Lawrence Berkeley Laboratory, MS: 50B - 2239, Berkeley, CA, 94720. Tel: 510-486-5014, fax: 510-486-6363; or Brian Tierney, Tel: 510-486-7381.
2. This work is jointly supported by ARPA - CSTO, and by the U. S. Dept. of Energy, Energy Research Division, Office of Scientific Computing, under contract DE-AC03-76SF00098 with the University of California. This document is LBL report LBL-35408. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement or recommendation by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes. The following terms are acknowledged as trademarks: UNIX (Novell, Inc.), Sun and SPARCStation (Sun Microsystems, Inc.), DEC and Alpha (Digital Equipment Corp.), SGI and Indigo (Silicon Graphics, Inc.).

Multimedia 94- 10/94 San Francisco, CA, USA
0-89791-686-7/94/0010

Mode (ATM) networking technology, due to the architecture of the SONET infrastructure that will underlie large scale ATM networks of the future, will provide the bandwidth that will enable the approach of using ATM network-based distributed, parallel data servers to provide high-speed, scalable storage systems.

The approach described here differs in many ways from RAID, and should not be confused with it. RAID is a particular data strategy used to secure reliable data storage and parallel disk operation. Our approach, while using parallel disks and servers, deliberately imposes no particular layout strategy, and is implemented entirely in software (though the data redundancy idea of RAID might be usefully applied across servers to provide reliability in the face of network problems).

## Overview

The Image Server System (ISS) is an implementation of a distributed parallel data storage architecture. It is essentially a "block" server that is distributed across a wide area network to supply data to applications located anywhere in the network. See Figure *1: Parallel Data and Server Architecture Approach to the Image Server System*. There is no inherent organization to the blocks, and in particular, they would never be organized sequentially on a server. The data organization is determined by the application as a function of data type and access patterns, and is implemented during the data load process. The usual goal of the data organization is that data is declustered (dispersed in such a way that as many system elements as possible can operate simultaneously to satisfy a given request) across both disks and servers. This strategy allows a large collection of disks to seek in parallel, and all servers to send the resulting data to the application in parallel, enabling the ISS to perform as a high-speed image server.

The functional design strategy is to provide a high-speed "block" server, where a block is a unit of data request and storage. The ISS essentially provides only one function - it responds to requests for blocks. However, for greater efficiency and increased usability, we have attempted to identify a limited set of functions that extend the core ISS functionality while allowing support for a range of applications. First, the blocks are "named." In other words, the view from an application is that of a *logical* block server. Second, block requests are in the form of lists that are taken by the ISS to be in priority order. Therefore the ISS attempts (but does not guarantee) to return the higher priority blocks first. Third, the application interface provides the ability to ascertain certain configuration parameters (e.g., disk server names, performance, disk configuration, etc.) in order to permit parameterization of block placement-strategy algorithms (for example, see [1]). Fourth, the ISS is instrumented to permit monitoring of almost every aspect of its functioning during operation. This monitoring functionality is designed to facilitate performance tuning and network performance research; however, a data layout algorithm might use this facility to determine performance parameters.

At the present state of development and experience, the ISS that we describe here is used primarily as a large, fast "cache". Reliability with respect to data corruption is provided only by the usual OS and disk mechanisms, and data delivery reliability of the overall system is a function of user-level strategies of data replication. The data of interest (tens to hundreds of GBytes) is typically loaded onto the ISS from archival tertiary storage, or written into the system from live video sources. In the latter case, the data is also archived to bulk storage in real-time.

## Client Use

The client-side (application) use of the ISS is provided through a library that handles initialization (for example, an "open" of a data set requires discovering all of the disk servers with which the application will have to communicate), and the basic block request / receive interface. It is the responsibility of the client (or its agent) to maintain information about any higher-level organization of the data blocks, to maintain sufficient local buffering so that "smooth playout" requirements may be met locally, and to run predictor algorithms that will pre-request blocks so that application response time requirements can be met.

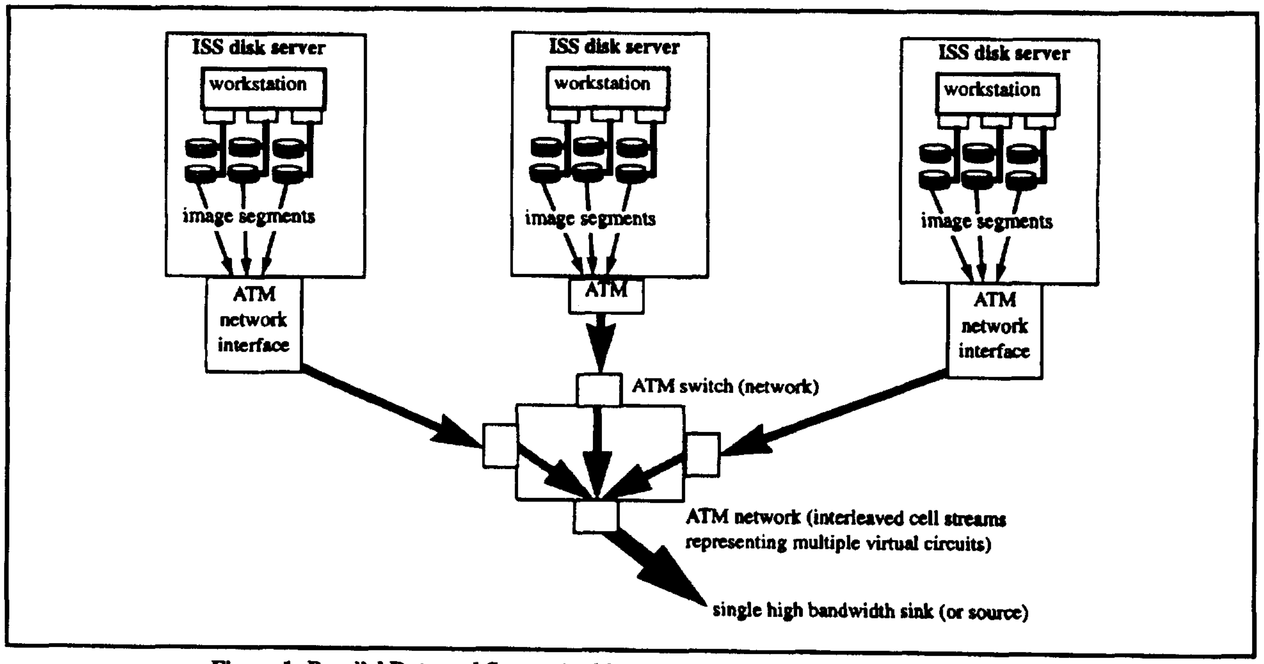


**Figure 1: Parallel Data and Server Architecture Approach to the Image Server System**

None of this has to be explicitly visible to the user-level application, but some agent in the client environment must deal with these issues, because the ISS always operates on a best-effort basis: if it did not deliver a requested block in the expected time or order, it was because it was not possible to do so.

**Implementation**

In our prototype implementations, the typical ISS consists of several (four - five) UNIX workstations (e.g. Sun SPARCStation, DEC Alpha, SGI Indigo, etc.), each with several (four - six) fast-SCSI disks on multiple (two - three) SCSI host adaptors. Each workstation is also equipped with an ATM network interface. An ISS configuration such as this can deliver an aggregated data stream to an application at about 400 Mbits/s (50 Mbytes/s) using these relatively low-cost, "off the shelf" components by exploiting the parallelism provided by approximately five servers, twenty disks, ten SCSI host adaptors, and five network interfaces.

Prototypes of the ISS have been built and operated in the MAGIC[3] network testbed. In this paper we describe mainly architecture and approach, as well as optimization strategies. A previous paper [11] describes the major implementation issues, and a paper to be published [12] will describe other ISS applications and ISS performance issues.

## 2.0 Related Work

There are other research groups working on solving problems related to distributed storage and fast multimedia data retrieval. For example, Ghandeharizadeh, Ramos, et al., at USC are working on declustering methods for multimedia data [2], and Rowe, et al., at UCB are working on a continuous media player based on the MPEG standard [10].

In some respects, the ISS resembles the Zebra network file system, developed by John H. Hartman and John K. Ousterhout at the University of California, Berkeley [3]. Both the ISS and Zebra can separate their data access and management activities across several hosts on a network. Both try to maintain the availability of the system as a whole by building in some redundancy, allowing for the possibility that a disk or host might be unavailable at a critical time. The goal of both is to increase data throughput despite the current limits on both disk and host throughput.

However, the ISS and the Zebra network file system differ in the fundamental nature of the tasks they perform. Zebra is intended to provide traditional file system functionality, ensuring the consistency and correctness of a file system whose contents are changing from moment to moment. The ISS, on the other hand, tries to provide very high-speed, high-throughput access to a relatively static set of data. It is optimized to retrieve data, requiring only minimum overhead to verify data correctness and no overhead to compensate for corrupted data.

## 3.0 Applications

There are several target applications for the initial implementation of the ISS. These applications fall into two categories: image servers and multimedia / video file servers.

### 3.1 Image Server

The initial use of the ISS is to provide data to a terrain visualization application in the MAGIC testbed. This application, known as TerraVision [5], allows a user to navigate through and over a high resolution landscape represented by digital aerial images and elevation models. TerraVision is of interest to the U.S. Army because of its ability to let a commander "see" a battlefield environment. TerraVision is very different from a typical "flight simulator"-like program in that it uses high resolution aerial imagery for the visualization instead of simulated terrain. TerraVision

3. MAGIC (Multidimensional Applications and Gigabit Internetwork Consortium) is a gigabit network testbed that was established in June 1992 by the U. S. Government's Advanced Research Projects Agency (ARPA)[9]. MAGIC's charter is to develop a high-speed, wide-area networking testbed that will demonstrate interactive exchange of data at gigabit-per-second rates among multiple distributed servers and clients using a terrain visualization application. More information about MAGIC may be found on the WWW home page at: http://www.magic.net/

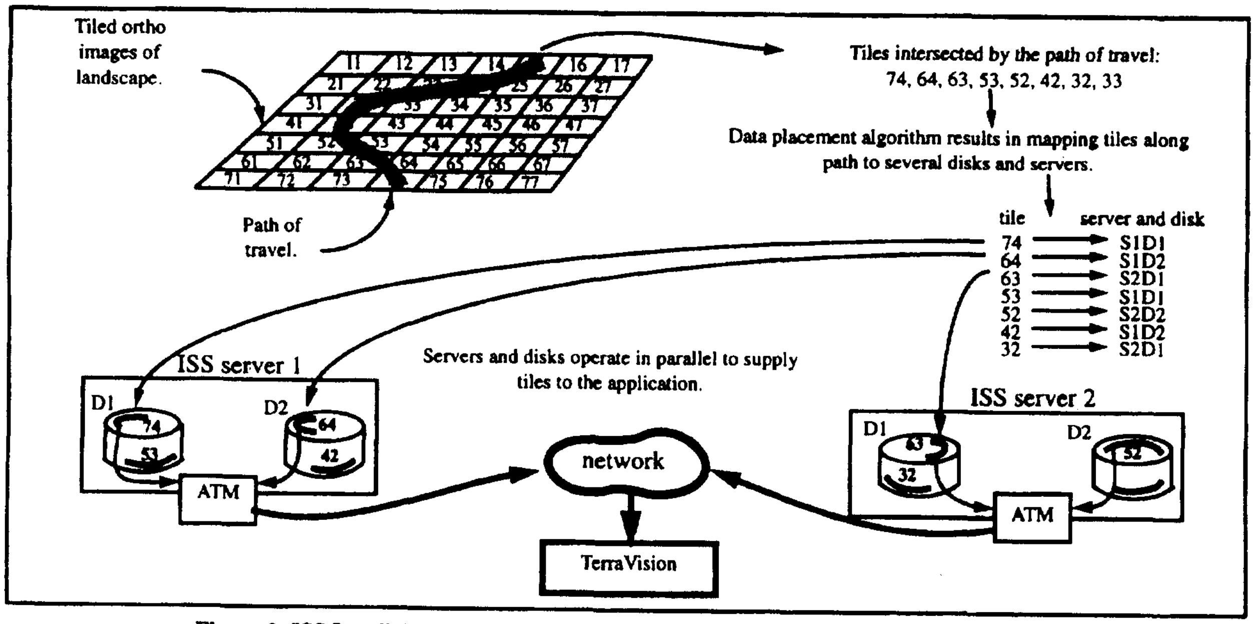


Figure 2: ISS Parallel Data Access Strategy as Illustrated by the TerraVision Application

requires large amounts of data, transferred at both bursty and steady rates. The ISS is used to supply image data at hundreds of Mbits/s rates to TerraVision. No data compression is used with this application because the bandwidth requirements are such that real-time decompression is not possible without using special purpose hardware.

In the case of a large-image browsing application like TerraVision, the strategy for using the ISS is straightforward: the image is tiled (broken into smaller, equal-sized pieces), and the tiles are scattered across the disks and servers of the ISS. The order of tiles delivered to the application is determined by the application predicting a "path" through the image (landscape), and requesting the tiles needed to supply a view along the path. The actual delivery order is a function of how quickly a given server can read the tiles from disk and send them over the network. Tiles will be delivered in roughly the requested order, but small variations from the requested order will occur. These variations must be accommodated by buffering, or other strategies, in the client application.

Figure 2: *ISS Parallel Data Access Strategy as Illustrated by the TerraVision Application* shows how image tiles needed by the TerraVision application are declustered across several disks and servers. More detail on this declustering is provided below.

Each ISS server is independently connected to the network, and each supplies an independent data stream into and through the network. These streams are formed into a single network flow by using ATM switches to combine the streams from multiple medium-speed links onto a single high-speed link. This high-speed link is ultimately connected to a high-speed interface on the visualization platform (client). On the client, data is gathered from buffers and processed into the form needed to produce the user view of the landscape.

This approach could supply data to any sort of large-image browsing application, including applications for displaying large aerial-photo landscapes, satellite images, X-ray images, scanning microscope images, and so forth.

Figure 3: *Use of the ISS for Single High-Bandwidth App.* shows how the network is used to aggregate several medium-speed streams into one high-speed stream for the image browsing application. For the MAGIC TerraVision application, the applica-

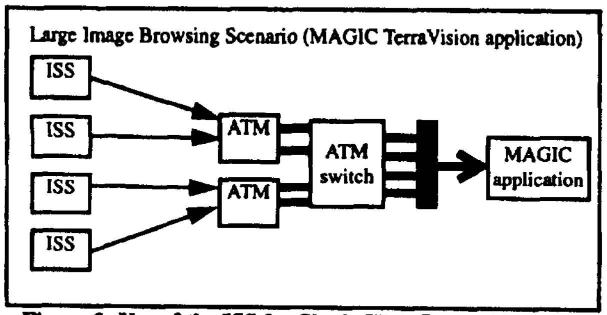

Large Image Browsing Scenario (MAGIC TerraVision application)
ISS
ISS
ISS
ISS
ATM
ATM
ATM switch
MAGIC application

**Figure 3: Use of the ISS for Single High-Bandwidth App.**

tion host (an SGI Onyx) is using multiple OC-3 (155 Mbit/s) interfaces to achieve the bandwidth requirements necessary. These multiple interfaces will be replaced by a single OC-12 (622 Mbit/s) interface when it becomes available.

In the MAGIC testbed, the ISS has been run in several ATM WAN configurations to drive several different applications, including TerraVision. The configurations include placing ISS servers in Sioux Falls, South Dakota (EROS Data Center), Kansas City, Kansas (Sprint), and Lawrence, Kansas (University of Kansas), and running the TerraVision client at Fort Leavenworth, Kansas (U. S. Army's Battle Command Battle Lab). The ISS disk server and the TerraVision application are separated by several hundred kilometers, the longest link being about 700 kilometers.

## 3.2 Video Server

Examples of video server applications include video players, video editors, and multimedia document browsers. A video server might contain several types of stream-like data, including conventional video, compressed video, variable time base video, multimedia hypertext, interactive video, and others. Several users would typically be accessing the same video data at the same time, but would be viewing different streams, and different frames in the same stream. In this case the ISS and the network are effectively being used to "reorder" segments (see Figure *4: Use of the ISS to Supply many Low-Bandwidth Streams*). This reordering

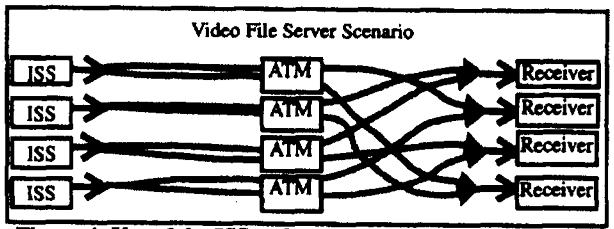


**Figure 4: Use of the ISS to Supply many Low-Bandwidth Streams**

affects many factors in an image server system, including the layout of the data on disks. Commercial concerns such as Time Warner and U.S. West are building large-scale commercial video servers such as the Time Warner / Silicon Graphics video server [4]. Because of the relatively low cost and ease of scalability of our approach, it may address a wider scale, as well as a greater diversity, of data organization strategies so as to serve the diverse needs of schools, research institutions, and hospitals for video-image servers in support of various educational and research-oriented digital libraries.

## 4.0 Design

### 4.1 Goals

The following are some of our goals in designing the ISS:

- The ISS should be capable of being geographically distributed. In a future environment of large scale, high-speed, mesh-connected national networks, network distributed storage should be capable of providing an uninterruptable stream of data, in much the same way that a power grid is resilient in the face of source failures, and tolerant of peak demands, because of the possibility of multiple sources multiply interconnected.
- The ISS approach should be scalable in all dimensions, including data set size, number of users, number of server sites, and aggregate data delivery speed.
- The ISS should deliver coherent image streams to an application, given that the individual images that make up the stream are scattered (by design) all over the network. In this case, "coherent" means "in the order needed by the application". No one disk server will ever be capable of delivering the entire stream. The network is the *server.*
- The ISS should be affordable. While something like a HIPPI-based RAID device might be able to provide functionality similar to the ISS, this sort of device is very expensive, is not scalable, and is a single point of failure.

## 4.2 Approach

### A Distributed, Parallel Server

The ISS design is based on the use of multiple low-cost, medium-speed disk servers which use the network to aggregate server output. To achieve high performance we exploit all possible levels of parallelism, including that available at the level of the disks, controllers, processors / memory banks, servers, and the network. Proper data placement strategy is also key to exploiting system parallelism.

At the server level, the approach is that of a collection of disk managers that move requested data from disk to memory cache. Depending on the nature of the data and its organization, the disk managers may have a strategy for moving other closely located and related data from disk to memory. However, in general, we have tried to keep the implementation of data prediction (determining what data will be needed in the near future) separate from the basic data-moving function of the server. Prediction might be done by the application (as it is in TerraVision), or it might be done be a third party that understands the data usage patterns. In any event, the server sees only lists of requested blocks.

As explained in [12], the dominant bottlenecks for this type of application in a typical UNIX workstation are first memory copy speed, and second, network access speed. For these reasons, an important design criterion is to use as few memory copies as possible, and to keep the network interface operating at full bandwidth all the time. Our implementation uses only three copies to get data from disk to network, so maximum server throughput is about (memory_copy_speed / 3).

Another important aspect of the design is that all components are instrumented for timing and data flow monitoring in order to characterize ISS and network performance. To do this, all communications between ISS components are timestamped. In the MAGIC testbed, we are using GPS (Global Positioning System) receivers and NTP (Network Time Protocol) [6] to synchronize the clocks of all ISS servers and of the client application in order to accurately measure network throughput and latency.

### Data Placement Issues

A limiting factor in handling large data sets is the long delay in managing and accessing subsets of these data sets. Slow I/O rates, rather than processor speed, are chiefly the cause of this delay. One way to address this problem is to use data reorganization techniques based on the application's view of the structure of the data, analysis of data access patterns, and storage device characteristics. By matching the data set organization with the intended use of the data, substantial improvements can be achieved for common patterns of data access[1]. This technique has been applied to large climate-modeling data sets, and we are applying it to TerraVision data stored in the ISS. For image tile data, the placement algorithm declusters tiles so that all disks are evenly accessed by tile requests, but then clusters tiles that are on the same disk based on the tiles' relative nearness to one another in the image. This strategy is a function of both the data structure (tiled images) and the geometry of the access (e.g., paths through the landscape).

The declustering method used for tiles of large images is a lattice-based (i.e., vector-based) declustering scheme, the goal of which is to ensure tiles assigned to the same server are as far apart as possible on the image plane. This minimizes the chance that the same server will be accessed many times by a single tile request list.

Tiles are distributed among K disks by first determining a pair of integer component vectors which span a parallelogram of area K. Tiles assigned to the same disk are separated by integer multiples of these vectors. Mathematical analysis shows that for common visualization queries this declustering method performs within seven percent of optimal for a wide range of practical multiple disk configurations.

Within a disk, however, it is necessary to cluster the tiles such that tiles near each other in 2-D space are close to each other on disk, thus minimizing disk seek time. The clustering method used here is based on the Hilbert Curve because it has been shown to be the best curve that preserves the 2-D locality of points in a 1-D traversal.

### Path Prediction

Path prediction is important to ensure that the ISS is utilized as efficiently as possible. By using a strategy that always requests more tiles than the ISS can actually deliver before the next tile request, we can ensure that no component of the ISS is ever idle. For example, if most of a request list's tiles were on one server, the other servers could still be reading and sending or caching tiles that may be needed in the future, instead of idly waiting. The goal of path prediction is to provide a rational basis for pre-requesting tiles. See [1] for more details on data placement methods.

As a simple example of path prediction, consider an interactive video database with a finite number of distinct paths (video clips), and therefore a finite number of possible branch points. (A "branch point" occurs where a user might select one of several possible play clips, see Figure *5: Image Stream Management / Prediction Strategy*). As a branch point is approached by the player, the predictor (without knowledge of which branch will be taken) will start requesting images (frames) along both branches. These images are cached first at the disk servers, then at the receiving application. As soon as a branch is chosen, the predictor ceases to send requests for images from the other branches. Any "images" (i.e., frames or compressed segments) cached on the ISS, but unsent, are flushed as better predictions fill the cache.

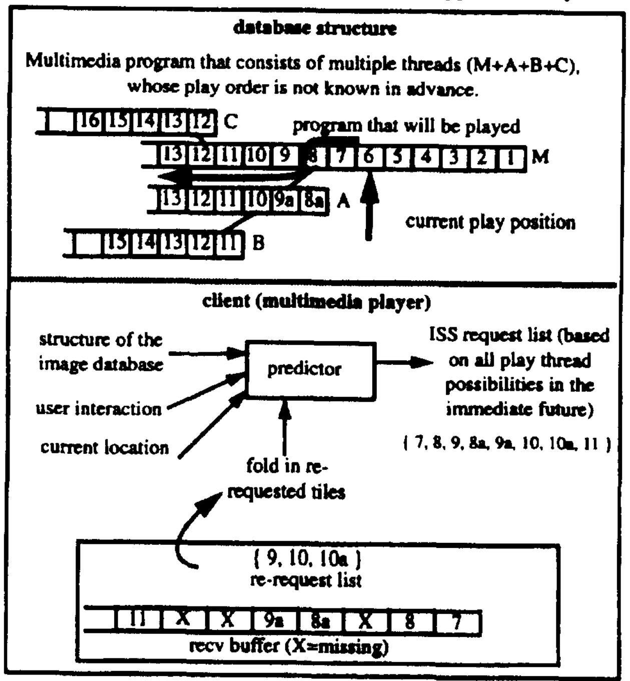

database structure
Multimedia program that consists of multiple threads (M+A+B+C), whose play order is not known in advance.
16 15 14 13 12 C
program that will be played
13 12 11 10 9 8 7 6 5 4 3 2 1 M
13 12 11 10 9a 8a A
current play position
15 14 13 12 11 B
client (multimedia player)
structure of the image database
user interaction
current location
predictor
ISS request list (based on all play thread possibilities in the immediate future)
{ 7, 8, 9, 8a, 9a, 10, 10a, 11 }
fold in re-requested tiles
{ 9, 10, 10a }
re-request list
11 X X 9a 8a X 8 7
recv buffer (X=missing)

Figure 5: Image Stream Management / Prediction Strategy

This is an example where a relatively independent third party might do the prediction.

The client will keep asking for an image until it shows up, or until it is no longer needed (e.g., in TerraVision, the application may have "passed" the region of landscape that involves the image that was requested, but never received.) Applications will have different strategies to deal with images that do not arrive in time. For example, TerraVision keeps a local, low-resolution, data set to fill in for missing tiles.

Prediction is transparent to the ISS, and is manifested only in the order and priority of images in the request list. The prediction algorithm is a function of the client application, and typically runs on the client.

### The Significance of ATM Networks

The design of the ISS depends in part on the ability of ATM switches and networks to aggregate multiple data streams from the disk servers into a single high-bandwidth stream to the application. This is feasible because most wide area ATM network aggregate bandwidth upward - that is, the link speeds tend to increase from LANs to WANs, and even within WANs the "backbone" is the highest bandwidth. (This is actually a characteristic of the architecture of the SONET networks that underlie ATM networks.) Aggregation of stream bandwidth occurs at switch output ports. For example, three incoming streams of 50 Mbits/s that are all destined for the same client will aggregate to a 150 Mbit/s stream at the switch output port. The client has data stream connections open to each of the ISS disk servers, and the incoming data from all of these streams typically put data into the same buffer.

## 5.0 Implementation

In a typical example of ISS operation the application sends requests for data (images, video, sound, etc.) to the name server process which does a lookup to determine the location (server/disk/offset) of the requested data. Requests are sorted on a per-server basis, and the resulting lists are sent to the individual servers. Each server then checks to see if the data is already in its cache, and if not, fetches the data from disk and transfers it to the cache. Once the data is in the cache, it is sent to the requesting application. Figure 6: *ISS Architecture* shows how the components

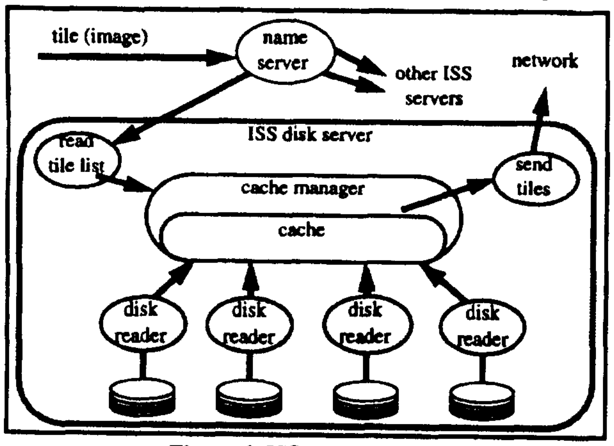


**Figure 6: ISS Architecture**

of the ISS are used to handle requests for data blocks.

The disk server handles three image request priority levels:

- high: send first, with an implicit priority given by order within the list.
- medium: send if there is time.
- low: fetch into the cache if there is time, but don't send.

The priority of a particular request is set by the requesting application. The application's prediction algorithm can use these priority levels to keep the ISS fully utilized at all times without requesting more data than the application can process. For example, the application could send low priority requests to pull data into the ISS cache, knowing that the ISS would not send the data on to the application until the application was ready. Another example is an application that plays back a movie with a sound track, where audio might be high priority requests, and video medium priority requests.

### 5.1 Performance Limits

Using a Sun SPARCStation 10-41 with two Fast-SCSI host adaptors and four disks, and reading into memory random 48 Kbyte tiles from all disks simultaneously, we have measured a single server disk-to-memory throughput of 9 Mbytes/s. When we add a process which sends UDP packets to the ATM interface, this reduces the disk-to-memory throughput to 8 Mbytes/s (64 Mbits/s). The network throughput under these conditions is 7.5 Mbytes/s (60 Mbits/s). This number is an upper limit on performance for this platform; it does not include the ISS overhead of buffer management, semaphore locks, and context switching. The SCSI host adaptor and Sbus are not yet saturated, but adding more disks will not help the overall throughput without faster access memory and to the network (e.g., multiple interfaces and multiple independent data paths as are used in systems like a SPARCServer 1000 or SGI Challenge).

## 6.0 Current Status

All ISS software is currently tested and running on Sun workstations (SPARCstations and SPARCserver 1000's) running SunOS 4.1.3 and Solaris 2.3, DEC Alpha's running OSF/1, and SGI's running IRIX 5.x. Demonstrations of the ISS with the MAGIC Terrain Visualization application TerraVision have been done using several WAN configurations in the MAGIC testbed [9]. Using enough disks (4-8, depending on the disk and system type), the ISS software has no difficulty saturating current ATM interface cards. We have worked with 100 Mbit and 140Mbit TAXI S-Bus and VME cards from Fore systems, and OC-3 (155 Mbit/s) cards from DEC, and in all cases ISS throughput is only slightly less than *ttcp*[4] speeds.

Table 1 below shows various system *ttcp* speeds and ISS speeds. The first column is the maximum *ttcp* speeds using TCP

**TABLE 1.**

| System | Max ATM LAN *ttcp* | *ttcp* w/ disk read | Max ISS speed |
|---|---|---|---|
| Sun SS10-51 | 70 Mbits/sec | 60 Mbits/sec | 55 Mbits/sec |
| Sun SS1000 (2 processors) | 75 Mbits/sec | 65 Mbits/sec | 60 Mbits/sec |
| SGI Challenge L (2 processors) | 82 Mbits/sec | 72 Mbits/sec | 65 Mbits/sec |
| Dec Alpha | 127 Mbits/sec | 95 Mbits/sec | 88 Mbits/sec |

4. *ttcp* is a utility that times the transmission and reception of data between two systems using the UDP or TCP protocols.

over a ATM LAN with a large TCP window size. In this case, *ttcp* just copies data from memory to the network. For the values in the second column, we ran a program that continuously reads from all ISS disks simultaneously with *ttcp* operation. This gives us a much more realistic value for what network speeds the system is capable or while the ISS is running. The last column is the actual throughput values measured from the ISS. These speeds indicate that the ISS software adds a relatively small overhead in terms of maximum throughput.

### 6.1 Actual Performance

The current throughput of a single ISS server on a Sun SPARC 10/41 platform is 7.1 Mbytes/s (55 Mbits/s), or 91% of the possible maximum of 7.5 Mbytes/s (60 Mbits/s) derived above. This seems a reasonable result considering the overhead required. We have achieved this speed using a TerraVision-like application simulator which we developed that sends a list of requests for data at a rate of five request lists per second. Five request lists per second does not force the application to predict and buffer too far into the future, but is not so fast that disk read latency is an issue. This application simulator sends request lists that are long enough to ensure that no disk ever is idle. When the ISS receives a request list, all previous requests are discarded. Under these conditions, about one-half of the requests in each request list will never be satisfied (either they will be read into the cache but not written to the network, or they will not be read at all before the next request list arrives).

As an example, a typical TerraVision request list contains fifty tiles. Of these fifty tiles, forty are read into ISS cache, twenty-five are written to the network, and ten are not processed at all. This behavior is reasonable because, as discussed in the section on data path prediction above, the application will keep asking for data until it shows up or is no longer needed. The requesting application will anticipate this behavior, and predict the tiles it needs far enough ahead that "important" tiles are always received by the time they are needed. Tiles are kept in the cache on an LRU basis, and previously requested but unsent tiles will be found in the cache by a subsequent request. The overhead of re-requesting tiles is minimal compared with moving them from disk and sending them over the network.

During ISS operation, the average CPU usage on the disk server platform is 10% user, 60% system, 30% idle, so the CPU is not a bottleneck. With the TerraVision application and 40 Mbyte of disk cache memory on the ISS server, on average 2% of tiles are already in cache relative to any given request. Increasing the cache size will not increase the throughput, but may improve latency with effective path prediction by the application.

## 7.0 Future Work

We plan to expand the capabilities of the ISS considerably during the next year or so. These enhancements (and associated investigation of the issues) will include:

- Implementing a multiple data set data layout strategy;
- Implementing a multi-user data layout and access strategy;
- Implementing a capability to write data to the ISS;
- Implementing the ability to monitor the state of all ISS servers and dynamically assign bandwidth of individual servers to avoid overloading the capacity of a given segment of the network (i.e., switches or application host);
- Implementing mechanisms for handling video-like data, including video data placement algorithms and the ability to handle variable size frames (JPEG/MPEG);
- Modifying name server design to accommodate data on server performance and availability and to provide a mechanism to request tiles from the "best" server (fastest or least loaded);
- Investigating the issues involved in dealing with data other than image- or video- like data.

Many of these enhancements will involve extensions to the data placement algorithm and the cache management methods. Also we plan to explore some optimization techniques, including using larger disk reads, and conversion of all buffer and device management processes to threads-based light weight processes.

## 8.0 References

[1] Chen L. T. and Rotem D., "Declustering Objects for Visualization", Proc. of the 19th VLDB (Very Large Database) Conference, 1993.

[2] Ghandeharizadeh, S. and Ramos, L, "Continuous Retrieval of Multimedia Data Using Parallelism, IEEE Transactions on Knowledge and Data Engineering, Vol 5, No 4, August 1993.

[3] Hartman, J. H. and Ousterhout, J. K., "Zebra: A Striped Network File System", Proceedings of the USENIX Workshop on File Systems, May 1992.

[4] Langberg, M., "Silicon Graphics Lands Cable Deal with Time Warner Inc.", San Jose Mercury News, June 8, 1993.

[5] Leclerc, Y.G. and Lau, S.Q., Jr.,"TerraVision: A Terrain Visualization System", SRI International, Technical Note #540, Menlo Park, CA, 1994.

[6] Mills, D., "Simple Network Time Protocol (SNTP)", RFC 1361, University of Delaware, August 1992.

[7] Parvin, B., Peng, C., Johnston, W., and Maestre, M.,"Tracking of Tubular Objects for Scientific Applications", IEEE Conf. on Computer Vision and Pattern Recognition, June 1994, pp. 295-301

[8] Patterson, D., Gibson, R., and Katz, R., "The Case for RAID: Redundant Arrays of Inexpensive Disks", Proceedings ACM SIGMOD Conference, Chicago, IL, May, 1988 (pp. 106-113)

[9] Richer, I and Fuller, B.B, "An Overview of the MAGIC Project," M93B0000173, The MITRE Corp., Bedford, MA, 1 Dec. 1993.

[10] Rowe L. and Smith B.C, "A Continuous Media Player", Proc. 3rd International Workshop on Network and Operating System Support for Digital Audio and Video, San Diego, CA, Nov. 1992.

[11] Tierney, B., Johnston, W., Herzog, H., Hoo, G., Jin, G., Lee, J., "System Issues in Implementing High Speed Distributed Parallel Storage Systems", Proceedings of the USENIX Symposium on High Speed Networking, Aug. 1994, LBL-35775.

[12] Tierney, B., Johnston, W., Chen, L.T., Herzog, H., Hoo, G., Jin, G., Lee, J., "Using High Speed Networks to Enable Distributed Parallel Image Server Systems", Proceedings of Supercomputing '94, Nov. 1994, LBL-35437.